REMARKS

Claims 1-33 are pending. Applicant has proposed to amend independent claims 1, 9, 19 and 27.

Applicants appreciate the allowance of claims 32 and 33 and the indication of allowable subject matter in claims 16 and 17.

Claims 1, 2, 4, 5, 9, 10, 12, 13, 19, 20, 22, 23 and 29 stand rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,463,582 ("Lethin"). Claims 3, 21 and 28 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Lethin in view of U.S. Patent No. 5,581,778 ("Chin"). Claims 6-8, 14, 15, 18, 24-26, 30 and 31 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Lethin in view of either U.S. Patent No. 6,397,379 ("Yates") or U.S. Patent No. 6,317,872 ("Gee"). Reconsideration is respectfully requested for the following reasons.

There does not appear to be any stated ground of rejection of independent claim 27.

This application has been pending for over six years. The new ground of rejection set forth in the present Office Action is the fifth new ground of rejection that has been raised in this case. As with the previous grounds of rejection, Applicants respectfully submit that the newly cited art does not teach or suggest the claimed invention.

As Applicants have explained in numerous prior responses, the present invention takes a novel approach to emulating the execution of a computer program comprising object code instructions of the native machine instruction set of a target (first) computer on a host (second) computer having a different native machine instruction set. This novel approach involves two steps. First, the object code instructions of the target program are statically translated into a series of instructions of an intermediate instruction set. The intermediate instruction set is optimized for interpretation on the second computer. The intermediate instruction set is not the native machine instruction set of either the first computer or the second computer. Second, the series of instructions of the intermediate instruction set are then directly interpreted on the second computer using an interpreter to emulate execution of the program; no further translation into the native machine (object) code of the second computer is performed. These features are recited in each of the independent claims 1, 9, 19, and 27. For example, claim 1 recites:

> performing a *static translation* of the object code instructions of the target program into a series of instructions of an intermediate instruction set *that is not the native machine instruction set of the host computer*, the intermediate instruction set being optimized for interpretation on the host computer; and *thereafter*,
>
> executing the series of instructions of the intermediate instruction set *by interpretation on the host computer using an interpreter*.

Neither the primary reference, Lethin, nor any of the secondary references appears to teach or suggest these features of the claimed invention. Indeed, Applicants respectfully submit that the portions of Lethin to which the Office Action refers in support of the Section 102(e) and 103(a) rejections are taken out of context.

Lethin describes a system that provides "a *dynamic* translation of a computer program in one machine language into another machine language *while the program is running*." Col. 1, ll. 64-67 (emphasis added).[1] Lethin recognizes that "the conventional dynamic object code conversion method has a problem in that all objects, including seldom used objects, are converted", *i.e.*, the conventional method "fails to efficiently recognize objects which are executed plural times and thereby increases the time needed for conversion of the original object code while sacrificing efficiency." Col. 1, ll. 41-45. Lethin's system attempts to improve upon the conventional *dynamic* object code conversion method by using an interpreter in combination with a compiler to more efficiently *dynamically* compile the object code of a program to be emulated into the object code of a host computer. As Lethin explains,

> The present invention generally relates to an optimizing object code translator, hereinafter ("OOCT"), which performs dynamic compilation of a microprocessor instruction set as part of a computer architecture emulation system. . . .
>
> Performing meaningful compiler-type optimizations is only possible with knowledge of an instruction flow graph. . . .

1 *See also*, col. 1, ll. 8-13 ("the present invention relates to the art of dynamic object code translators which perform analysis and computation of an original object code instruction set in real time during execution on a host processor having a host processor object code instruction set") (emphasis added); and Abstract ("An optimizing object code translation system and method perform dynamic compilation and translation of a target object code on a source operating system while performing optimization. Compilation and optimization of the target code is dynamically executed in real time.") (emphasis added).

> [T]o determine the flow graph, the program must be run. An interpreter runs the program the first time. As the interpreter executes the program, the interpreter informs OOCT each time that it performs a branch operation. This logging of information identifies some of the instructions and some of the join points. . . .
>
> The dynamic compilation chooses which portions of the text to optimize based on profiling information gathered by the interpreter.

Col. 4, ln. 45 – col. 5, ln. 23. Lethin elaborates on the manner in which the interpreter and compiler communicate later in the specification:

> Interpreter 110 and compiler 104 communicate with each other in several ways. The interpreter 110 records branch information into a branch log by communicating with branch logger 112. Compiler 104 is also able to read the branch log. Compiler 104 creates compiled code segments and stores their entry points in the Translation Table, which interpreter 110 reads.
> . . .
> Interpreter 110 transfers execution to compiled code when interpreter 110 calls a branch logging routine and it finds a compiled code segment for the branch destination . . . .

Col. 6, ll. 40-46 and col. 21, ll. 54-57. Thus, Lethin describes a *dynamic* object code translator that uses an interpreter to provide "flow graph" information to enable code to be *dynamically* translated more efficiently. Lethin does not describe a system or method as claimed by Applicants in which a target program to be emulated on a host computer is *first statically translated* to a series of instructions of an intermediate instruction set optimized for interpretation on the host computer, and *then* the series of instructions of the intermediate instruction set are executed directly by interpretation on the host computer.

The Office Action refers to column 50, lines 12-17 in support of its assertion that Lethin teaches static translation of the instructions of a target program into an intermediate instruction set optimized for interpretation on the host computer. Office Action, ¶ 4 at p. 2. But this portion of the specification is merely pointing out that existing "servers available for static translation of Java source code into Java bytecode" do not have the same advantages that Lethin's *dynamic* compiler provides. Specifically, Lethin states:

> There are also translation servers available for static translation of Java source code into Java bytecode [3]. These servers offer

> the advantages of a separate compiler task for static translation *but not for dynamic translation*, because they do not operate while the Java program is running.

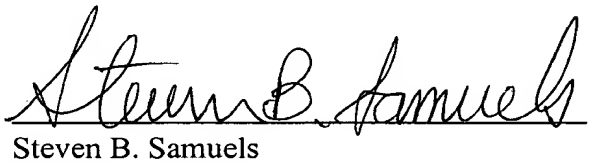
Col. 50, ll. 12-17 (emphasis added). Thus, this portion of the specification is **not** describing any aspect of the Lethin system, but rather is distinguishing the Lethin system from certain existing translation servers.

When the cited portions of Lethin are considered in context, it is readily apparent that those portions of Lethin do not teach or suggest a system or method like that claimed by Applicants in which a target program to be emulated on a host computer is *first statically translated* to a series of instructions of an intermediate instruction set optimized for interpretation on the host computer, and *then* the series of instructions of the intermediate instruction set are executed directly by interpretation on the host computer. Consequently, Lethin does not anticipate any of the claims of the instant application, nor does it provide the base teachings necessary to support any of the Section 103(a) rejections of the claims. Moreover, none of the secondary references – Chin, Yates, or Gee – cures the deficiencies of Lethin. For these reasons, Applicants respectfully request reconsideration of the Section 102(e) and 103(a) rejections of claims 1-31.

CONCLUSION

For all of the foregoing reasons, Applicants respectfully submit that the instant application is in condition for allowance.

Respectfully submitted,



Date: November 14, 2005

Steven B. Samuels
Registration No. 37,711

Woodcock Washburn LLP
One Liberty Place - 46th Floor
Philadelphia PA 19103
Telephone: (215) 568-3100
Facsimile: (215) 568-3439